09/885081

Patent Application of
Kevin W Jameson
For

COLLECTION COMMAND APPLICATOR

CROSS REFERENCES TO RELATED APPLICATIONS

The present invention uses inventions from the following patent applications, that are filed contemporaneously herewith, and which are incorporated herein by reference:

USPTO 09/885078, Collection Information Manager; Kevin Jameson.
USPTO 09/885080, Collection Recognizer; Kevin Jameson.

FIELD OF THE INVENTION

This invention relates to automated software systems for processing collections of computer files in arbitrary ways, thereby improving the productivity of software developers, web media developers, and other humans and computer systems that work with collections of computer files.

Collection information managers are software modules that obtain and organize collection information from collection information stores into information-rich collection data structures that are used by application programs.

Collection Physical Representations -- Main Embodiment

Figures 1-3 show the physical form of a simple collection, as would be seen on a personal computer filesystem.

FIG 1 shows an example prior art filesystem folder from a typical personal computer filesystem. The files and directories shown in this drawing do not implement a collection 100, because no collection specifier 102, FIG 2 Line 5 exists to associate a collection type definition FIG 4 101 with collection content information FIG 4 103.

09/885,081

FIG 2 shows the prior art folder of FIG 1, but with a portion of the folder converted into a collection 100 by the addition of a collection specifier file FIG 2 Line 5 named "cspec". In this example, the collection contents FIG 4 103 of collection 100 are defined by two implicit policies of a preferred implementation.

First is a policy to specify that the root directory of a collection is a directory that contains a collection specifier file. In this example, the root directory of a collection 100 is a directory named "c-myhomepage" FIG 2 Line 4, which in turn contains a collection specifier file 102 named "cspec" FIG 2 Line 5.

Second is a policy to specify that all files and directories in and below the root directory of a collection are part of the collection content. Therefore directory "s" FIG 2 Line 6, file "homepage.html" FIG 2 Line 7, and file "myphoto.jpg" FIG 2 Line 8 are part of collection content FIG 4 103 for said collection 100.

FIG 3 shows an example physical representation of a collection specifier file 102, FIG 2 Line 5, such as would be used on a typical personal computer filesystem.

09/885,081

Collection Information Types

Figures 4-5 show three kinds of information that comprise collection information.

FIG 4 shows a high-level logical structure of three types of information that comprise collection information: collection processing information 101, collection specifier information 102, and collection content information 103. A logical collection 100 is comprised of a collection specifier 102 and collection content 103 together. This diagram best illustrates the logical collection information relationships that exist within a preferred filesystem implementation of collections.

FIG 5 shows a more detailed logical structure of the same three types of information shown in FIG 4. Collection type definition information FIG 4 101 has been labeled as per-type information in FIG 5 103 because there is only one instance of collection type information 101 per collection type. Collection content information FIG 4 103 has been labeled as per-instance information in FIG 5 103 because there is only one instance of collection content information per collection instance. Collection specifier information 102 has been partitioned into collection instance processing information 104, collection-type link information 105, and collection content link information 106. FIG 5 is intended to show several important types of information 104-106 that are contained within collection specifiers 102.

Suppose that an application program means FIG 6 110 knows (a) how to obtain collection processing information 101, (b) how to obtain collection content information 103, and (c) how to relate the two with per-collection-instance information 102. It follows that application program means FIG 6 110 would have sufficient knowledge to use collection processing information 101 to process said collection content 103 in useful ways.

Collection Command Applicator Means

FIG 11 shows a simplified architecture for a collection command applicator (CCA) program. A CCA manager module 120 oversees the command application process.

Module Get Runtime Info 121 obtains input arguments from the invocation command line, obtains runtime option values and configuration values from the execution environment, and otherwise prepares initial data for the command application process.

Module Collection List Producing Means 140 oversees the process of obtaining a list of target collections to which commands will be applied. Several different methods of obtaining the list are described below. The obtained list typically contains the identities of target collections, filesystem locations of target collections, and other useful collection information about the target collections.

10/585081

Module Command Execution Means 160 oversees the process of applying commands to the list of target collections. Module Command Execution Sequential Means 161 applies commands to collections in sequential order, such that a command application to one collection is completed before another command application to another collection begins. In contrast, Module Command Execution Parallel Means 162 applies commands to collections in parallel, such that a single command can be applied to many collections in parallel. Parallel application of commands is useful because it reduces the time required to perform an entire command application to a set of target collections.

Operation

In operation, CCA Manager 120 proceeds according to the simplified algorithm shown in FIG 12.

First, CCA Manager 120 calls Get Runtime Info 121 to obtain runtime information and load it into a data structure "runtime-info" such as shown in FIG 13. In particular, Get Runtime Info 121 is responsible for ensuring that commands that are to be applied FIG 13 Line 5 are present within the data structure for later use by Command Execution Means 160.

Next, CCA Manager 120 calls Collection List Producing Means 140 to obtain a list of target collections and associated collection information for subsequent command application.

Finally, CCA Manager 120 calls Command Execution Means 160 to apply commands obtained by Get Runtime Info 121 to each collection in the list of target collections, thereby completing the command application function of the CCA program.

Now that overall program structure and operation have been described, more detailed explanations can be usefully provided below.

Collection List Producing Means

FIG 14 shows an expanded architecture for the Collection List Producing Means 140 shown in FIG 11. The collection list producing mechanism performs two main functions: obtaining a list of target collections, and sorting the list of target collections.

First, Get Collection List 141 is responsible for obtaining a list of target collections for command application. Several different methods of calculating a list are possible, including the methods represented by modules 142-144.

Get Collection List Explicit 142 obtains an explicit collection list from information provided to the CCA program invocation via Get Runtime Info 121. FIG 20 shows an list of collection recognizer information Line 6, and other information of interest to the CCA program.

Finally, data structure "coll-list-prod-info" FIG 18 is returned by Collection List Producing Means 140 to the calling module CCA Manager 120, for eventual use in command application.

Command Execute Sequential Means

FIG 21 shows an expanded architecture for the Command Execute Sequential Means 161 shown in FIG 11. Two sequential command execution approaches are possible: direct and indirect.

In direct execution mode, applied commands are executed by a CCA program in real time, such that all command applications are completed before the CCA program invocation is completed.

09/885281

In contrast, an indirect command execution approach does not use real time execution. Instead, a CCA program using an indirect command execution approach generates an output script file FIG 26 that can be reused many times to apply commands to target collections. Script files contain a specific list of all target collections identified at the time the file was created, but typically do not contain any commands to apply. Rather, script files contain internal argument placeholders for receiving commands to apply from the command line. This approach enables script files to be reused many times, each time with differing commands provided on the script invocation command line.

In practice, indirect command application is very useful because it saves the repetitive cost of recalculating the same target collection set each time a new command must be applied to a stable working set of collections. Moreover, generated script files are tangible files that can be moved, copied, stored, reused, and otherwise

Next, Command Execute Sequential Direct Means 170 traverses the list of target collections stored in the data structure "all-coll-cmd-exe" FIG 25. List traversal begins at FIG 22 Line 4.

For each collection in the list of target collections, a subordinate module FIG 21 171-173 is called to perform a command application using a desired execution method. The particular type of execution method (fork, thread,...) is specified by runtime information such as command line arguments or CCA program configuration options.

Command execution status information is collected and stored into "cmd-exe-status" FIG 23 data structures as algorithm FIG 22 proceeds. Implementation policies control behavior if a command application fails. That is, execution may continue to the next target collection in the list, or may be aborted for the whole CCA program invocation, as policies dictate.

Once command application is complete, data structure "all-coll-cmd-exe" FIG 25 is returned by Command Execute Sequential Direct 170 to Command Execute Sequential Means 161 and eventually to Command Execute Means 160 and CCA Manager 120.

Command execution results are extracted from data structures and communicated to CCA users. The type and quantity of results information returned is determined by implementation policy. Typically, normal execution results are printed or displayed in real time as commands are executed, and a final program status code is returned to the operating system when the CCA program completes.

Command Execute Sequential Indirect Means

Command Execute Sequential Indirect Means 180 is generally responsible for indirectly and sequentially executing commands on the list of target collections.

collection signatures is defined by implementation policy.

Collection selection proceeds by comparing each detected collection with provided selection criteria. Selection criteria can be complex, and may be based on collection instance data (within the collection specifier), collection type data (within the collection type definition), or collection content information (within files that belong to the collection). Typical collection selections are based on collection specifier information. More complex searches are possible, and would use collection type information or collection content information. The combined process of detection and selection is called collection recognition.

Information obtained from a recognition process is stored into a "rec-coll" data structure FIG 29 as recognition proceeds. The main element of the recognized collections data structure is a list of recognized collections FIG 29 Line 3, heavily augmented with additional information FIG 29 Lines 5-8 about each collection in the list.

09/885081

Once recognition has completed, a recognized collections data structure FIG 29 representing the search is returned by Collection Recognizer Means 143 to Get Collection List 141, and eventually to Collection List Producing Means 140 and CCA Manager 120, for subsequent use in command application.

Collection Recognizers play a very important, very practical role in enabling the construction of scalable, automated command application systems.

Collection Visit Ordering Means

A second important enhanced embodiment is concerned with solving the collection visit order problem. That is, the problem of applying commands to collections in accordance with execution-order interdependencies among the target collections. For example, CCA invocations that implement software builds must calculate and use a including shell sorts, tree sorts, insertion sorts, and quicksorts. The particular choice of sorting algorithm is determined by implementation policy.

After determining a numeric visit order value for each target collection on the unsorted list, Visit Order Means 152 sorts the unsorted collections by numeric visit order using a chosen sorting algorithm. Sort results for each sort are stored in a data structure "sorted-colls" FIG 17. A list of "sorted-colls" FIG 17 data structures is stored in a "coll-list-prod-info" FIG 18 Line 5 data structure, which could support multiple sortings if such were desired. In typical practice, however, only one visit order sort is used.

Once sorting is complete, a "coll-list-prod-info" FIG 18 data structure containing a list of target collections sorted by execution visit order is returned by Collection List Producing Means Manager 140 to the calling module CCA Manager 120, for eventual use in applying commands.

Continuing, the sorted list of collections FIG 18 is passed into Command Execution Means 160 for use in either direct or indirect command applications.

For direct command applications, Command Execute Sequential Direct Means 161 would visit collections according to the visit orderings stored in the sorted lists of collections FIG 18 Line 5. By way of example, FIG 33 shows an example unsorted visit order sequence for the collection tree of FIG 30. In contrast, FIG 35 shows the same collections sorted into correct execution visit order sequence, using the visit order sorting techniques described above.

For indirect command applications, Command Execute Sequential Indirect Means 162 would use a sorted list of collections to emit properly ordered command sequences into a script file. By way of example, FIG 36 shows a script file that uses proper execution visit ordering to visit collections in the collection tree of FIG 30. Note that the script file visits collections in proper execution visit order, the same

09/828527

Operation

In operation, Collection List Producing Means 140 proceeds according to the simplified algorithm shown in FIG 15. Generally speaking, Collection List Producing Means 140 calls Get Collection List 141 to obtain a list of target collections, and then calls Sort Collection List 150 to sort the list of collections.

First, the algorithm builds data structures FIG 16-18 to support the pending computation.

To construct a list of target collections FIG 15 Lines 3-6, Get Collection List 141 calls one or more of its subordinate modules 142-144 to obtain collection lists, in accordance with command line control arguments provided to the invocation. In a simple preferred embodiment, Get Collection Explicit 142 could be called to produce a list, and could use a simple text file FIG 20 to explicitly list collection pathnames within a particular collection tree FIG 19. In this simple preferred embodiment, the text file FIG 20 could be provided as a command line argument to the CCA program invocation. Alternative means 143-144 of constructing a list are also possible, such as by using a Collection Recognizer Means 143. But since Collection Recognizer Means 143 is a more complex (but preferred) method of building a collection list, it is described later in this document as an enhanced embodiment.

09/885087

The obtained collection list is stored in a data structure "target-coll-list" FIG 16. This data structure is essentially a list of smaller individual collection data structures FIG 8. A collection data structure FIG 8, when fully populated, contains essentially all there is to know about a collection, with the exception that actual collection content is not stored within the data structure.

Once a list of collections has been obtained, data structure "target-coll-list" FIG 16 is returned by Get Collection List 141 to the calling module Collection List Producing

181	Generate sequential script file
182	Generate sequential program file
183	Execute sequential indirect other means

200	Command execute parallel direct means
201	Calculate parallel execution groups
202	Execute parallel fork means
203	Execute parallel thread means
204	Execute parallel direct other means

210	Command execute parallel indirect means
212	Generate parallel script file
213	Generate parallel program file
214	Execute parallel indirect other means

09/885081

DETAILED DESCRIPTION

Overview of Collections

This section introduces collections and some related terminology.

Collections are sets of computer files that can be manipulated as a set, rather than as individual files. Collection information is comprised of three major parts: (1) a collection specifier that contains information about a collection instance, (2) a collection type definition that contains information about how to process all collections of a particular type, and (3) optional collection content in the form of arbitrary computer files that belong to a collection.